Chapter 14. Installable Device Drivers

Contents

Introduction

The DOS Version 2.00 device interface links the devices together in a chain. This allows new device drivers for optional devices to be added to DOS.

Device Driver Format

A device driver is a .COM file with all of the code in it to implement the device. In addition it has a special header at the front of it that identifies it as a device, defines the strategy and interrupt entry points, and defines various attributes of the device.

> Note: For device drivers, the .COM file must not use the ORG 100H. Because it does not use the program segment prefix, the device driver is simply loaded; therefore, the .COM file must have an orgin of zero (ORG 0 or no ORG statement).

Types of Devices

There are two basic types of devices:

- Character devices
- Block devices

Character Devices

These are devices that are designed to do character I/O in a serial manner like CON, AUX, and PRN. These devices have names like CON, AUX, CLOCK$, and you can open channels (handles or FCBs) to do input and output to them.

Note: Because character devices have only one name, they can support only one device.

Block Devices

These devices are the "fixed disk or diskette drives" on the system, they can do random I/O in pieces called blocks (usually the physical sector size of the disk). These devices are not *named* as the character devices are, and cannot be "opened" directly. Instead they are *mapped* via the drive letters (A, B, C, etc.). Block devices can have units within them. In this way, a single block driver can be responsible for one or more disk or diskette drives. For example, block device driver ALPHA can be responsible for drives A, B, C and D. This means that ALPHA has four units defined and therefore takes up four drive letters. The way the drive units and drive letters correspond is determined by the position of the driver in the chain of all drivers. For example, if device driver ALPHA is the first block driver in the device chain, and it has defined four units, then those units will be A, B, C and D. If BETA is the second block driver, and it defines three units, then those units will be E, F and G. DOS Version 2.00 is not limited to 16 block device units as previous versions were. The new limit is 63, but drives are assigned alphabetically through the collating sequence, so after drive Z, the drive "characters" get a little strange (like <,\ , >).

Device Header

A device header is required at the beginning of a device driver. Here is what the Device Header looks like:

Description	Definition
Pointer to next device header	DWORD
Attribute	WORD
Pointer to device strategy	WORD
Pointer to device interrupt	WORD
Name/unit field	8 BYTES

Next Device Header Field

The pointer to the next device header field is a double word field (offset followed by segment) that is set by DOS at the time the device driver is loaded. However, it is important that this field be set to -1 prior to load time (when it is on the disk as a .COM file) unless there is more than one device driver in the .COM file. If there is more than one driver in the file, the first word of the double word pointer should be the offset of the next driver's Device Header.

Note: If there is more than one device driver in the .COM file, the *last* driver in the file must have the pointer to next Device Header field set to -1.

Attribute Field

The next field in the header describes to the system the attributes of the device. They are as follows:

bit 15 = 1 if character device
0 if block device

bit 14 = 1 if IOCTL is supported
0 if it is not

bit 13 = 1 if non IBM format (block only)
0 if IBM format

bit 3 = 1 if current clock device
0 if it is not

bit 2 = 1 if current NUL device
0 if it is not

bit 1 = 1 if current standard output device
0 if it is not

bit 0 = 1 if current standard input device
0 if it is not

All other bits must be off.

The most important bit is bit 15, which tells the system that it is a block or a character device. With the exception of bits 13 and 14, the rest are for giving character devices special treatment and mean nothing on a block device. These *special treatment* bits allow you to tell DOS that your new device driver is the new standard input device and standard output device (the CON device). This can be done by setting bits 0 and 1 to 1. Similarly, a new CLOCK$ device could be installed by setting that attribute bit.

Although there is a NUL device attribute bit, the NUL device *cannot be reassigned.* This is an attribute that exists for DOS so it can tell if the NUL device is being used. The non IBM format bit applies only to block devices and affects the operation of the Get BPB (BIOS Parameter Block) device call (covered later in this chapter). The other bit of interest is the IOCTL bit. This is used for both block and character devices, and tells DOS whether the device is able to handle control strings (through the IOCTL system call).

If a driver cannot process control strings, it should initially set this bit to 0. This way DOS can return an error if an attempt is made through the IOCTL system call to send or receive control strings to the device. A device that is able to process such control strings should initialize this bit to 1. For devices of this type, DOS will make the calls to the IOCTL input and the IOCTL output device functions to send and receive IOCTL strings.

The IOCTL functions allow data to be sent to and from the device without actually doing a normal read or write. In this way, the device can use the data for its own use (like setting a baud rate, stop bits, changing form lengths, etc.). It is up to the device to interpret the information passed to it, but it must not be treated as a normal I/O request.

Strategy and Interrupt Routines

These two fields are the pointers to the entry points of the strategy and interrupt routines. They are word values, so they must be in the same segment as the Device Header.

Name Field

This is an 8-byte field that contains the name of a character device, or the number of units of a block device. If it is a block device, the number of units can be put in the first byte. This is optional, because DOS will fill in this location with the value returned by the driver's INIT code. (Refer to "Installation of Device Drivers" in this chapter.)

Creating a Device Driver

In order to create a device driver that DOS can install, a .COM file must be created with the Device Header at the start of the file. Remember that for device drivers, the code should not be originated at 100H, but rather at 0. The link field (pointer to next Device Header) should be –1 unless there is more than one device driver in the .COM file. The attribute field and entry points must be set correctly.

If it is a character device, the name field should be filled in with the name of that character device. The name can be any legal 8-character filename.

DOS always processes installable device drivers before handling the default devices, so to install a new CON device, simply name the device CON (just be sure to set the standard input device and standard output device bits in the attribute word on a new CON device). The scan of the device list stops on the first match, so the installable device driver takes precedence.

Note: Because DOS can install the driver anywhere in memory, care must be taken in any far memory references. You should not expect that your driver will always be loaded at the same place every time.

Installation of Device Drivers

DOS Version 2.00 allows new device drivers to be installed dynamically at boot time by reading and processing the device options in the CONFIG.SYS file.

DOS calls a device driver at it's strategy entry point first, passing in a Request Header the information describing what DOS wants the device driver to do.

The strategy routine does not perform the request, but rather it enqueues the request (saves a pointer to the Request Header). The second entry point is the interrupt routine, and is called by DOS immediately after the strategy routine returns. The "interrupt" routine is called with no parameters. Its function is to perform the operation based on the queued request and set up any return information.

DOS passes the pointer to the Request Header in ES:BX. This structure consists of a fixed length header (Request Header) followed by data pertinent to the operation to be performed.

> **Note:** It is the responsibility of the device driver to preserve the machine state (for example, save all registers on entry, and restore them on exit).
>
> The stack used by DOS will have enough room on it to save all of the registers. If more stack space is needed, it is the device drivers responsibility to allocate and maintain another stack.
>
> All calls to device drivers are FAR calls, and FAR returns should be executed to return to DOS. (See "Sample Device Driver" listing at the end of this chapter.)

Request Header

BYTE length in bytes of the Request Header plus any data at the end of the Request Header
BYTE unit code The subunit the operation is for (minor device). Has no meaning for character devices.
BYTE command code
WORD Status
8 BYTE area reserved for DOS
Data appropriate to the operation

Unit Code

The unit code field identifies which unit in your device driver the request is for. For example, if your device driver has 3 units defined, then the possible values of the unit code field would be 0, 1, and 2.

Command Code

The command code field in the Request Header can have the following values:

Code	Function
0	INIT
1	MEDIA Check (Block only, NOP for character)
2	BUILD BPB (Block only, NOP for character
3	IOCTL input (only called if IOCTL bit is 1)
4	INPUT (read)
5	NON-DESTRUCTIVE INPUT NO WAIT (Character devices only)
6	INPUT STATUS (Character devices only)
7	INPUT FLUSH (Character devices only)
8	OUTPUT (write)
9	OUTPUT (write) with verify
10	OUTPUT STATUS (Character devices only)
11	OUTPUT FLUSH (Character devices only)
12	IOCTL output (only called if IOCTL bit is 1)

BUILD BPB and MEDIA CHECK

BUILD BPB and MEDIA CHECK, for block devices only, are explained here.

DOS calls MEDIA CHECK first for a drive unit. DOS passes it's current Media Descriptor byte (see "Media Descriptor Byte" later in this chapter). MEDIA CHECK returns one of the following four results:

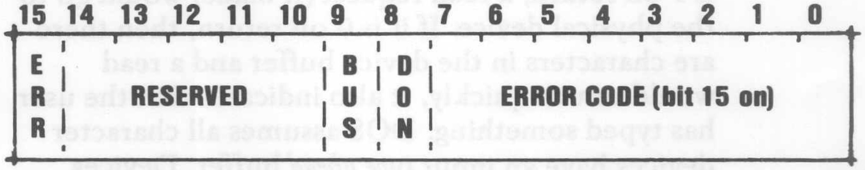
- Media Not Changed
- Media Changed
- Not Sure

DOS will call BUILD BPB under the following two conditions:

- If "Media Changed" is returned
- If "Not Sure" is returned and there are no dirty buffers (buffers with changed data, not yet written to disk).

Status Word

The status word in the request Header.



The status word is zero on entry and is set by the driver interrupt routine on return.

Bit 8 is the done bit. When set it means the operation is complete. For DOS 2.00 the Driver just sets it to one when it exits.

Bit 15 is the error bit. If it is set, then the low 8 bits of the status word indicate the error. The errors are:

00	Write Protect Violation
01	Unknown Unit
02	Device Not Ready
03	Unknown command
04	CRC Error
05	Bad Drive Request Structure Length
06	Seek Error
07	Unknown Media
08	Sector Not Found
09	Printer Out of Paper
0A	Write Fault
0B	Read Fault
0C	General Failure

Bit 9 is the busy bit that is set by status calls.

For output on character devices: If it is 1 on return, a write request (if made) would wait for completion of a current request. If it is 0, there is no current request, and a write request (if made) would start immediately.

For input on character devices with a buffer: If it is 1 on return, a read request (if made) would go to the physical device. If it is 0 on return, then there are characters in the device buffer and a read would return quickly, it also indicates that the user has typed something. DOS assumes all character devices have an input *type ahead* buffer. Devices that do not have them should always return busy = 0 so that DOS will not continuously wait for something to get into a buffer that does not exist.

One of the functions defined for each device is INIT. This routine is called only once when the device is installed and never again. There are several things returned by the INIT routine. First, there is a location of the first free byte of memory after the device driver (like a terminate and stay resident) that is stored in the ending address field. In this manner, initialization code can be used once and thrown away in order to save space.

After sending the ending address field, a character device driver can set the status word and return. While block devices are installed in the same way as character devices, they must return additional information. The manner of units for the device driver is returned, and this determines the logical names that the devices will have. For example, if the current maximum logical device letter is F at the time of the install call, and the block device driver INIT routine returns 3 units, then their logical names will be G, H, and I. This mapping is determined by the position of the driver in the device list, and the number of units on the device. The number of units returned by INIT will override the value in the name/unit field of the Device Header.

In addition, a pointer to a BPB (BIOS Parameter Block) pointer array is also returned. This is a pointer to an array of *n* word pointers, where *n* is the number of units defined. These word pointers point to BPBs. In this way, if all of the units are the same, the entire array can point to the same BPB in order to save space.

> Note: This array must be protected (below the free pointer set by the return).

The BPB (BIOS Parameter Block) contains information pertinent to the devices like sector size, sectors per allocation unit, etc.. The sector size in the BPB cannot be greater than the maximum allowed (set at DOS initialization time).

The last thing that INIT of a block device must pass back is the "media descriptor byte". This byte means nothing to DOS, but is passed to devices so that they know what parameters DOS is currently using for a particular Drive-Unit.

Block devices may take several approaches; they may be *dumb* or *smart*. A dumb drive would define a unit (and therefore a BPB) for each possible media drive combination. Unit 0 = drive 0 single side, unit 1 = drive 0 double side, etc. For this approach, media descriptor bytes would mean nothing. A smart device would allow multiple media per unit. In this case, the BPB table returned at INIT must define space large enough to accommodate the largest possible media supported (sector size in BPB must be as large as maximum sector size that DOS is currently using). Smart drivers will use the "media byte" to pass information about what media is currently in a unit.

Function Call Parameters

All strategy routines are called with ES:BX pointing to the Request Header. The interrupt routines get the pointers to the Request Header from the queue the strategy routines store them in. The command code in the Request Header tells the driver which function to perform.

Note: All DWORD pointers are stored offset first, then segment.

INIT

Command code=0

ES:BX

13-BYTE Request Header
BYTE number of units (not set by character device)
DWORD Ending Address
DWORD Pointer to BPB array (not set by Character devices)

The driver must do the following:

- Set the number of units (block devices only).
- Set up the pointer to the BPB array (block devices only).
- Perform any initialization code (to modems, printers etc.).
- Set up the ending address for resident code.
- **Set the status word in the Request Header.**

  Note: If there are multiple device drivers in a single .COM file, the ending address returned by the last INIT called will be the one DOS uses. For the sake of simplicity, it is recommended that all of the device drivers in a single .COM file return the same ending address.

MEDIA CHECK

Command code=1

ES:BX

13-BYTE Request Header
BYTE Media Descriptor from DOS
BYTE return information

The driver must perform the following:

- Set the return byte:

-1	Media has been changed
0	Don't know if media has been changed
1	Media has not been changed

- Set the status word in the Request Header.

BUILD BPB (BIOS Parameter Block)

Command code=2

ES:BX

13-BYTE Request Header
BYTE Media Descriptor from DOS
DWORD Transfer Address (buffer address)
DWORD Pointer to BPB table

The driver must perform the following:

- Set the pointer to the BPB.

- Set the status word in the Request Header.

The driver must determine the correct media that is currently in the unit to return the pointer to the BPB table. The way the buffer is used (pointer passed by DOS) is determined by the non-IBM format bit in the attribute field of the device header. If the bit is zero (device is IBM format campatible) then the buffer contains the first sector of the FAT (most importantly the FAT id byte). The driver must not alter this buffer in this case. If the bit is a one, then the buffer is a one sector scratch area that can be used for anything.

If the device is IBM format compatible, then it must be true that the first sector of the first FAT is located at the same sector for all possible media. This is because the FAT sector is read *before* the media is actually determined.

The information relating to the BPB for a particular media is kept in the boot sector for the media. In particular, the format of the boot sector is:

3 BYTE near JUMP to boot code	
8 BYTE OEM name and version	
WORD bytes per sector	BPB
BYTE sectors per allocation unit (must be a power of 2)	
WORD reserved sectors (starting at logical sector 0)	
BYTE number of FATs	
WORD number of root dir entries (maximum allowed)	
WORD number of sectors in logical image (total sectors in media, including boot sector, directories, etc.)	
BYTE media descriptor	
WORD number of sectors occupied by a single FAT	
WORD sectors per track	
WORD number of heads	
WORD number of hidden sectors	

The three words at the end are optional. DOS does not care about them because they are not part of the BPB. They are intended to help the device driver understand the media. Sectors per track may be redundant because it can be calculated from the total size of the disk. The number of heads is useful for supporting different multi-head drives that have the same storage capacity but a different number of surfaces. The number of hidden sectors is useful for supporting drive partitioning schemes.

MEDIA Descriptor Byte

Currently the media descriptor byte has been defined for a few media types:

Media descriptor byte —>	1	1	1	1	1	x	x	x
	7	6	5	4	3	2	1	0

Bit	Meaning	
0	1=2 sided	0=not 2 sided
1	1=8 sector	0=not 8 sector
2	1=removable	0=not removable
3–7	must be set to 1	

Examples of current DOS media descriptor bytes:

- 5 1/4" Diskettes:

hex FC	1 sided 9 sector
hex FD	2 sided 9 sector
hex FE	1 sided 8 sector
hex FF	2 sided 8 sector

- Fixed Disks:

hex F8 (Fixed disk)

- 8" Diskettes:

hex FE (IBM 3740 Format). Single sided, single density, 128 bytes per sector, soft sectored, 4 sectors per allocation unit, 1 reserved sector, 2 FATs, 68 directory entries, 77*26 sectors.

hex FD (IBM 3740 Format). Dual sided, single density, 128 bytes per sector, soft sectored, 4 sectors per allocation unit, 4 reserved sectors, 2 FATs, 68 directory entries, 77*26 sectors.

hex FE. Single sided, double density, 1024 bytes per sector, soft sectored, 1 sector per allocation unit, 1 reserved sector, 2 FATs, 192 directory entries, 77*8*2 sectors.

Note: The two MEDIA descriptor bytes that are the same for 8" diskettes (hex FE) is not a misprint. To establish whether a diskette is single density or double density, a read of a single density address mark should be made. If an error occurs, the media is double density.

INPUT or OUTPUT

Command codes=3,4,8,9, and 12

ES:BX

13-BYTE Request Header
BYTE Media descriptor byte
DWORD transfer address (buffer address)
WORD byte/sector Count
WORD starting sector number (no meaning on character devices)

The driver must perform the following:

- Do the requested function.
- Set the actual number of sectors (bytes) transferred.
- Set the status word in the Request Header.

  Note: No error checking is performed on an IOCTL call. However, the driver must set the return sector (byte) count to the correct number transferred.

The following applies to block device drivers:

Under certain circumstances the device driver may be asked to do a write operation of 64K bytes that seems to be a *wrap around* of the transfer address in the device driver request packet. This arises due to an optimization added to the write code in DOS. It will only happen on WRITEs that are within a sector size of 64K bytes on files that are being extended past the current end of file. It is allowable for the device driver to ignore the balance of the WRITE that wraps around, if it so choses. For example, a WRITE of 10000H bytes worth of sectors with a transfer address of xxxx:1 could ignore the last two bytes.

Remember: A program that uses DOS function calls can never request an input or output operation of more than FFFFH bytes; therefore, a wrap around in the transfer (buffer) segment cannot occur. It is for this reason that you can ignore bytes that would have wrapped around in the transfer segment.

Non Destructive Input No Wait

Command code=5

ES:BX

13-BYTE Request Header
BYTE read from device

The driver must perform the following:

- Return a byte from the device.
- Set the status word in the Request Header.

This call is analagous to the console input status call on previous versions of DOS. If the character device returns busy bit =0 (characters in buffer), then the next character that would be read is returned. This character is not removed from the input buffer (hence the term Non Destructive Input). This call allows DOS to look ahead one input character.

STATUS

Command codes=6 and 10

ES:BX

13-BYTE Request Header

All driver must do is perform the operation and set the status word in the Request Header accordingly.

FLUSH

Command codes=7 and 11

ES:BX

13-BYTE Request Header

This call tells the driver to flush (terminate) all pending requests that it has knowledge of. Its primary use is to flush the input queue on character devices. The driver must set status word in the Request Header upon return.

The CLOCK$ Device

A popular add on feature is a "Real Time Clock" board. To allow these boards to be integrated into the system for TIME and DATE, there is a special device (determined by the attribute word) which is the CLOCK$ device. In all respects, this device defines and performs functions like any other character device (most functions will be set done bit, reset error bit, return). When a read or write to this device occurs, exactly 6 bytes are transferred. The first two bytes are a word which is the count of days since 1-1-80. The third byte is minutes, the fourth hours, the fifth 1/100 seconds, and the sixth seconds. Reading the CLOCK$ device gets the date and time, writing to it sets the date and time.

```
                     ; ****************************************************
                     ; *                     PROLOG                       *
                     ; *  THIS IS AN INSTALLABLE DEVICE DRIVER FOR AN     *
                     ; *  IN STORAGE DISKETTE (VIRTUAL) WITH 180K CAPACITY. *
                     ; ****************************************************
 0000                CSEG    SEGMENT PARA PUBLIC 'CODE'
                     ;
                     ;          M A C R O ( S )
                     ;
                     STATUS  MACRO   STATE,ERR,RC
                             IFIDN   <STATE>,<DONE>
                              OR      ES:WORD PTR SRH_STA_FLD[BX],0100H
                             ENDIF
                             IFIDN   <STATE>,<BUSY>
                              OR      ES:WORD PTR SRH_STA_FLD[BX],0200H
                             ENDIF
                             IFIDN   <ERR>,<ERROR>
                              OR      ES:WORD PTR SRH_STA_FLD[BX],1000H
                             ENDIF
                             IFNB    <RC>
                              OR      ES:WORD PTR SRH_STA_FLD[BX],RC
                             ENDIF
                             ENDM
                     ;
                     ;          E Q U A T E S
                     ;
                     ; READ/WRITE
                     ;
 = 0000              SRH          EQU     0                 ;STATIC REQUEST HEADER START
 = 000D              SRH_LEN      EQU     13                ; "      "      "     LENGTH
 =                   SRH_LEN_FLD EQU      SRH               ; "      "      "       "    FIELD
 = 0001              SRH_UCD_FLD EQU      SRH+1             ; "      "      "     UNIT CODE FIELD
 = 0002              SRH_CCD_FLD EQU      SRH+2             ; "      "      "     COMMAND CODE FIELD
 = 0003              SRH_STA_FLD EQU      SRH+3             ; "      "      "     STATUS FIELD
 = 0005              SRH_RES_FLD EQU      SRH+5             ; "      "      "     RESERVED AREA FIELD
                     ;
 = 000D              MD           EQU     SRH+SRH_LEN       ;MEDIA DESCRIPTOR BYTE
 = 0001              MD_LEN       EQU     1                 ; "       "       "    LENGTH
 = 000E              DTA          EQU     MD+MD_LEN         ;DISK TRANSFER ADDRESS
 = 0004              DTA_LEN      EQU     4                 ; DTA LENGTH
 = 0012              COUNT        EQU     DTA+DTA_LEN       ;BYTE/SECTOR COUNT
 = 0002              COUNT_LEN    EQU     2                 ;    "        "  LENGTH
 = 0014              SSN          EQU     COUNT+COUNT_LEN   ;STARTING SECTOR NUMBER
 = 0002              SSN_LEN      EQU     2                 ;  "       "       "   LENGTH
                     ;
                     ; MEDIA CHECK
                     ;
 = 000E              RET_BYTE     EQU     MD+MD_LEN         ;BYTE RETURNED FROM DRIVER
                     ;
                     ; BUILD BPB
                     ;
 = 0012              BPBA_PTR      EQU    DTA+DTA_LEN       ;POINTER TO BPB
 = 0004              BPBA_PTR_LEN EQU     4                 ;  "    " "  LENGTH
                     ;
                     ; INIT
```

```
                                ;
 = 000D                         UNITS       EQU     SRH+SRH_LEN
 = 0001                         UNITS_LEN   EQU     1
 = 000E                         BR_ADDR_0   EQU     UNITS+UNITS_LEN
 = 0010                         BR_ADDR_1   EQU     BR_ADDR_0+2
 = 0004                         BR_ADDR_LEN EQU     4
 = 0012                         BPB_PTR_OFF EQU     BR_ADDR_0+BR_ADDR_LEN
 = 0014                         BPB_PTR_SEG EQU     BPB_PTR_OFF+2
                                ;
                                ;
 0000                           VDSK    PROC    FAR
                                        ASSUME  CS:CSEG,ES:CSEG,DS:CSEG
 0000                           BEGIN:
 = 0000                         START       EQU     $
                                ;        S P E C I A L   D E V I C E   H E A D E R
 0000  FF FF FF FF              NEXT_DEV    DD      -1             ;POINTER TO NEXT DEVICE
 0004  2000                     ATTRIBUTE   DW      2000H          ;BLOCK DEVICE (NON-IBM FORMAT)
 0006  00E1 R                   STRATEGY    DW      DEV_STRATEGY   ;POINTER TO DEVICE STRATEGY
 0008  00EC R                   INTERRUPT   DW      DEV_INT        ;POINTER TO DEVICE INTERRUPT HANDLER
 000A  01                       DEV_NAME    DB      1              ;NUMBER OF BLOCK DEVICES
 000B     07 [                              DB      7 DUP(?)       ;7 BYTES OF FILLER
                ??
                   ]


                                ;
 0012  ????                     RH_OFF      DW      ?              ;REQUEST HEADER OFFSET
 0014  ????                     RH_SEG      DW      ?              ;REQUEST HEADER SEGMENT
                                ; BIOS PARAMETER BLOCK
 = 0016                         BPB     EQU     $
 0016  0200                                 DW      512            ;SECTOR SIZE
 0018  01                                   DB      1              ;SECTORS/ALLOCATION UNIT
 0019  0001                                 DW      1              ;NUMBER OF RESERVED SECTORS
 001B  02                                   DB      2              ;NUMBER OF FATS
 001C  0040                                 DW      64             ;NUMBER OF DIRECTORY ENTRIES
 001E  0168                                 DW      360            ;TOTAL NUMBER OF SECTORS
 0020  FC                                   DB      0FCH           ;MEDIA DESCRIPTOR
 0021  0002                                 DW      2              ;NUMBER OF SECTORS OCCUPIED BY FAT
                                ;
 0023  0016 R                   BPB_PTR DW      BPB                ;BIOS PARAMETER BLOCK POINTER ARRAY (1 ENTRY)
                                ; CURRENT VIRTUAL DISK INFORMATION
 0025  ????                     TOTAL       DW      ?              ;TOTAL SECTORS TO TRANSFER
 0027  00                       VERIFY      DB      0              ;VERIFY  1=YES, 0=NO
 0028  0000                     START_SEC   DW      0              ;STARTING SECTOR NUMBER
 002A  0000                     VDISK_PTR   DW      0              ;STARTING SEGMENT OF VIRTRUAL DISK
 002C  ????????                 USER_DTA    DD      ?              ;POINTER TO CALLERS DISK TRANSFER ADDRESS
 = 0030                         BOOT_REC    EQU     $              ;DUMMY DOS BOOT RECORD
 0030     03 [                              DB      3 DUP(0)       ;3 BYTE JUMP TO BOOT CODE (NOT BOOTABLE)
                00
                   ]

 0033  49 42 4D 20 20 32                    DB      'IBM  2.0'     ;VENDOR IDENTIFICATION
       2E 30
 003B  0200                                 DW      512            ;NUMBER OF BYTES IN A SECTOR
 003D  01                                   DB      1              ;1 SECTOR PER ALLOCATION UNIT
 003E  0001                                 DW      1              ;1 RESERVED SECTOR
```

```
0040 02                             DB     2              ;2 FATS
0041 0040                           DW     64             ;NUMBER OF DIRECTORY ENTRIES
0043 0168                           DW     360            ;360 TOTAL SECTORS IN IMAGE
0045 FC                             DB     0FCH           ;TELLS DOS THIS IS A SINGLE SIDED 9 SECTOR D
0046 0002                           DW     2              ;NUMBER OF SECTORS IN FAT
                         ;
                         ;  FUNCTION TABLE
                         ;
0048                     FUNTAB     LABEL  BYTE
0048 0105 R                         DW     INIT           ;INITIALIZATION
004A 01B8 R                         DW     MEDIA_CHECK    ;MEDIA CHECK (BLOCK ONLY)
004C 01CC R                         DW     BUILD_BPB      ;BUILD BPB       "      "
004E 0212 R                         DW     IOCTL_IN       ;IOCTL INPUT
0050 0212 R                         DW     INPUT          ;INPUT (READ)
0052 0212 R                         DW     ND_INPUT       ;NON_DESTRUCTIVE INPUT NO WAIT (CHAR ONLY)
0054 0212 R                         DW     IN_STAT        ;INPUT STATUS            "      "
0056 0212 R                         DW     IN_FLUSH       ;INPUT FLUSH             "      "
0058 0241 R                         DW     OUTPUT         ;OUTPUT (WRITE)
005A 0280 R                         DW     OUT_VERIFY     ;OUTPUT (WRITE) WITH VERIFY
005C 0212 R                         DW     OUT_STAT       ;OUTPUT STATUS           "      "
005E 0212 R                         DW     OUT_FLUSH      ;OUTPUT FLUSH            "      "
0060 0212 R                         DW     IOCTL_OUT      ;IOCTL OUTPUT
                         ;
                         ;L O C A L   P R O C E D U R E S
                         ;
0062                     IN_SAVE PROC  NEAR
0062 26: 8B 47 0E               MOV    AX,ES:WORD PTR DTA[BX] ;SAVE CALLERS DTA
0066 2E: A3 002C R              MOV    CS:USER_DTA,AX
006A 26: 8B 47 10               MOV    AX,ES:WORD PTR DTA+2[BX]
006E 2E: A3 002E R              MOV    CS:USER_DTA+2,AX
0072 26: 8B 47 12               MOV    AX,ES:WORD PTR COUNT[BX] ;GET NUMBER OF SECTORS TO READ
0076 32 E4                      XOR    AH,AH
0078 2E: A3 0025 R              MOV    CS:TOTAL,AX             ;MOVE NUMBER OF SECTORS TO TOTAL
007C C3                         RET
007D                     IN_SAVE ENDP
                         ;
007D                     CALC_ADDR PROC NEAR
007D 2E: A1 0028 R              MOV    AX,CS:START_SEC     ;GET STARTING SECTOR NUMBER
0081 B9 0020                    MOV    CX,20H              ;MOV 512 TO CX SEGMENT STYLE
0084 F7 E1                      MUL    CX                  ;MULTIPLY TO GET ACTUAL SECTOR
0086 2E: 8B 16 002A R           MOV    DX,CS:VDISK_PTR     ;GET SEGMENT OF VIRTUAL DISK
008B 03 D0                      ADD    DX,AX               ;ADD THAT SEGMENT TO INITIAL SEGMENT
008D 8E DA                      MOV    DS,DX               ;SAVE THAT AS THE ACTUAL SEGMENT
008F 33 F6                      XOR    SI,SI               ;IT'S ON A PARAGRAPH BOUNDARY
0091 2E: A1 0025 R              MOV    AX,CS:TOTAL         ;TOTAL NUMBER OF SECTORS TO READ
0095 B9 0200                    MOV    CX,512              ;BYTES PER SECTOR
0098 F7 E1                      MUL    CX                  ;MULTIPLY TO GET COPY LENGTH
009A 0B C0                      OR     AX,AX               ;CHECK FOR GREATER THAN 64K
009C 75 03                      JNZ    MOVE_IT
009E B8 FFFF                    MOV    AX,0FFFFH           ;MOVE IN FOR 64K
00A1                     MOVE_IT:
00A1 91                         XCHG   CX,AX               ;MOVE LENGTH TO CX
00A2 C3                         RET
00A3                     CALC_ADDR ENDP
                         ;
```

```
00A3                    SECTOR_READ PROC NEAR
00A3  E8 007D R                 CALL    CALC_ADDR               ;CALCULATE THE STARTING "SECTOR"
00A6  2E: 8E 06 002E R          MOV     ES,CS:USER_DTA+2        ;SET DESTINATION (ES:DI) TO POINT
00AB  2E: 8B 3E 002C R          MOV     DI,CS:USER_DTA          ;TO CALLERS DTA

                        ; CHECK FOR DTA WRAP IN CASE WE CAME THROUGH VIA VERIFY
                        ;
00B0  8B C7                     MOV     AX,DI                   ;GET OFFSET OF DTA
00B2  03 C1                     ADD     AX,CX                   ;ADD COPY LENGTH TO IT
00B4  73 07                     JNC     READ_COPY               ;CARRY FLAG = 0, NO WRAP
00B6  B8 FFFF                   MOV     AX,0FFFFH               ;MAX LENGTH
00B9  2B C7                     SUB     AX,DI                   ;SUBTRACT DTA OFFSET FROM MAX
00BB  8B C8                     MOV     CX,AX                   ;USE THAT AS COPY LENGTH TO AVOID WRAP
00BD                    READ_COPY:
00BD  F3/ A4                    REP     MOVSB                   ;DO THE "READ"
00BF  C3                        RET
00C0                    SECTOR_READ ENDP
                        ;
00C0                    SECTOR_WRITE PROC NEAR
00C0  E8 007D R                 CALL    CALC_ADDR               ;CALCULATE STARTING "SECTOR"
00C3  1E                        PUSH    DS
00C4  07                        POP     ES                      ;ESATABLISH ADDRESSABILITY
00C5  8B FE                     MOV     DI,SI                   ; ES:DI POINT TO "DISK"
00C7  2E: 8E 1E 002E R          MOV     DS,CS:USER_DTA+2        ; DS:SI POINT TO CALLERS DTA
00CC  2E: 8B 36 002C R          MOV     SI,CS:USER_DTA
                        ;
                        ; CHECK FOR DTA WRAP
                        ;
00D1  8B C6                     MOV     AX,SI                   ;MOVE DTA OFFSET TO AX
00D3  03 C1                     ADD     AX,CX                   ;ADD COPY LENGTH TO OFFSET
00D5  73 07                     JNC     WRITE_COPY              ;CARRY FLAG = 0, NO SEGMENT WRAP
00D7  B8 FFFF                   MOV     AX,0FFFFH               ;MOVE IN MAX COPY LENGTH
00DA  2B C6                     SUB     AX,SI                   ;SUBTRACT DTA OFFSET FROM MAX
00DC  8B C8                     MOV     CX,AX                   ;USE AS NEW COPY LENGTH TO AVOID WRAP
00DE                    WRITE_COPY:
00DE  F3/ A4                    REP     MOVSB                   ;DO THE "WRITE"
00E0  C3                        RET
00E1                    SECTOR_WRITE ENDP
```

```
                                       PAGE
                              ;
                              ; D E V I C E   S T R A T E G Y
                              ;
 00E1                         DEV_STRATEGY:
 00E1  2E: 8C 06 0014 R                MOV     CS:RH_SEG,ES           ;SAVE SEGMENT OF REQUEST HEADER POINTER
 00E6  2E: 89 1E 0012 R                MOV     CS:RH_OFF,BX           ;SAVE OFFSET  OF    "       "      "
 00EB  CB                              RET
                              ;
                              ; D E V I C E   I N T E R R U P T   H A N D L E R
                              ;
 00EC                         DEV_INT:
                              ; PRESERVE MACHINE STATE ON ENTRY
 00EC  FC                              CLD
 00ED  1E                              PUSH    DS
 00EE  06                              PUSH    ES
 00EF  50                              PUSH    AX
 00F0  53                              PUSH    BX
 00F1  51                              PUSH    CX
 00F2  52                              PUSH    DX
 00F3  57                              PUSH    DI
 00F4  56                              PUSH    SI
                              ;
                              ; DO THE BRANCH ACCORDING TO THE FUNCTION PASSED
                              ;
 00F5  26: 8A 47 02                    MOV     AL,ES:[BX]+2           ;GET FUNCTION BYTE
 00F9  D0 C0                           ROL     AL,1                   ;GET OFFSET INTO TABLE
 00FB  8D 3E 0048 R                    LEA     DI,FUNTAB              ;GET ADDRESS OF FUNCTION TABLE
 00FF  32 E4                           XOR     AH,AH
 0101  03 F8                           ADD     DI,AX
 0103  FF 25                           JMP     WORD PTR[DI]
                              ;
                              ; INIT
                              ;
 0105                         INIT:
 0105  0E                              PUSH    CS
 0106  5A                              POP     DX                     ;CURRENT CS TO DX
 0107  2E: 8D 06 02A0 R                LEA     AX,CS:VDISK            ;GET ADDRESS OF VIRTUAL DISK
 010C  B1 04                           MOV     CL,4
 010E  D3 C8                           ROR     AX,CL                  ;DIVIDE BY 16 (PARAGRAPH FORM)
 0110  03 D0                           ADD     DX,AX                  ;ADD TO CURRENT CS VALUE
 0112  2E: 89 16 002A R                MOV     CS:VDISK_PTR,DX        ;SAVE AS STARTING SEGMENT OF VIRTUAL DISK
 0117  B8 2D00                         MOV     AX,2D00H               ; ADD 2D00H PARAGRAPHS TO STARTING
 011A  03 D0                           ADD     DX,AX                  ;      SEGMENT OF VIRTUAL DISK
 011C  26: C7 47 0E 0000               MOV     ES:WORD PTR BR_ADDR_0[BX],0
 0122  26: 89 57 10                    MOV     ES:BR_ADDR_1[BX],DX    ;MAKE THAT THE BREAK ADDRESS
 0126  26: C6 47 0D 01                 MOV     ES:BYTE PTR UNITS[BX],1 ;NUMBER OF DISKETTE UNITS
 012B  8D 16 0023 R                    LEA     DX,BPB_PTR             ;GET ADDRESS OF BPB POINTER ARRAY
 012F  26: 89 57 12                    MOV     ES:BPB_PTR_OFF[BX],DX  ;SAVE OFFSET IN DATA PACKET
 0133  26: 8C 4F 14                    MOV     ES:BPB_PTR_SEG[BX],CS  ;SAVE SEGMENT IN DATA PACKET
 0137  2E: 8E 06 002A R                MOV     ES,CS:VDISK_PTR        ;GET STARTING SECTOR OF VIRTUAL DISK
 013C  33 FF                           XOR     DI,DI                  ;ZERO OUT DI (BOOT RECORD)
 013E  8D 36 0030 R                    LEA     SI,BOOT_REC            ;ADDRESS OF BOOT RECORD
 0142  B9 0018                         MOV     CX,24                  ;
 0145  F3/ A4                 REP      MOVSB                          ;COPY 24 BYTES OF BOOT RECORD
```

Drivers

```
0147  2E: C7 06 0028 R 0001            MOV      CS:WORD PTR START_SEC,1
014E  2E: C7 06 0025 R 0002            MOV      CS:WORD PTR TOTAL,2
0155  E8 007D R                        CALL     CALC_ADDR               ;CALCULATE ADDRESS OF LOGICAL SECTOR 1
0158  1E                               PUSH     DS
0159  07                               POP      ES
015A  8B FE                            MOV      DI,SI                   ;MOVE THAT ADDRESS TO ES:DI
015C  32 C0                            XOR      AL,AL
015E  F3/ AA                   REP     STOSB                            ;ZERO OUT FAT AREA
0160  C6 04 FC                         MOV      DS:BYTE PTR [SI],0FCH   ;SET THE FIRST FAT ENTRY
0163  C6 44 01 FF                      MOV      DS:BYTE PTR 1[SI],0FFH
0167  C6 44 02 FF                      MOV      DS:BYTE PTR 2[SI],0FFH
016B  1E                               PUSH     DS                      ;SAVE POINTER TO FAT
016C  56                               PUSH     SI                      ;              ON THE STACK
016D  2E: C7 06 0028 R 0003            MOV      CS:WORD PTR START_SEC,3
0174  2E: C7 06 0025 R 0002            MOV      CS:WORD PTR TOTAL,2
017B  E8 007D R                        CALL     CALC_ADDR               ;CALCULATE ADDRESS OF LOGICAL SECTOR 3
017E  1E                               PUSH     DS
017F  07                               POP      ES
0180  8B FE                            MOV      DI,SI                   ;MOVE THAT ADDRESS TO ES:DI
0182  5E                               POP      SI
0183  1F                               POP      DS                      ;RESTORE ADDRESS TO FIRST FAT
0184  F3/ A4                   REP     MOVSB                            ;COPY FIRST FAT TO SECOND FAT
0186  2E: C7 06 0028 R 0005            MOV      CS:WORD PTR START_SEC,5
018D  2E: C7 06 0025 R 0004            MOV      CS:WORD PTR TOTAL,4
0194  E8 007D R                        CALL     CALC_ADDR               ;CALCULATE ADDR OF L.S. 5 (START OF DIR)
0197  32 C0                            XOR      AL,AL
0199  1E                               PUSH     DS
019A  07                               POP      ES                      ;SET UP ES:DI TO POINT TO IT
019B  33 FF                            XOR      DI,DI
019D  F3/ AA                   REP     STOSB                            ;ZERO OUT DIRECTORY
019F  2E: 8E 06 0014 R                 MOV      ES,CS:RH_SEG            ;RESTORE ES:BX TO REQUEST HEADER
01A4  2E: 8B 1E 0012 R                 MOV      BX,CS:RH_OFF
                                       STATUS   DONE,NOERROR,0          ;SET STATUS WORD (DONE, NOERROR)
01A9  26: 81 4F 03 0100     +           OR       ES:WORD PTR SRH_STA_FLD[BX],0100H
01AF  26: 81 4F 03 0000     +           OR       ES:WORD PTR SRH_STA_FLD[BX],0
01B5  E9 0289 R                        JMP      EXIT
                                     ;
                                     ; MEDIA CHECK
                                     ;
01B8                                 MEDIA_CHECK:                       ;MEDIA CHECK (BLOCK ONLY)
                                     ;
                                     ;  SET MEDIA NOT CHANGED
                                     ;
01B8  26: C6 47 0E 01                  MOV      ES:BYTE PTR RET_BYTE[BX],1 ;STORE IN RETURN BYTE
                                       STATUS   DONE,NOERROR,0          ;TURN ON THE DONE BIT
01BD  26: 81 4F 03 0100     +           OR       ES:WORD PTR SRH_STA_FLD[BX],0100H
01C3  26: 81 4F 03 0000     +           OR       ES:WORD PTR SRH_STA_FLD[BX],0
01C9  E9 0288 R                        JMP      EXIT
                                     ;
                                     ; BUILD BIOS PARAMETER BLOCK
                                     ;
01CC                                 BUILD_BPB:
01CC  06                               PUSH     ES                      ;SAVE SRH SEGMENT
01CD  53                               PUSH     BX                      ;SAVE SRH OFFSET
01CE  2E: C7 06 0028 R 0000            MOV      CS:WORD PTR START_SEC,0
```

```
01D5  2E: C7 06 0025 R 0001             MOV      CS:WORD PTR TOTAL,1
01DC  E8 007D R                         CALL     CALC_ADDR               ;CALCULATE ADDRESS OF FIRST SECTOR
01DF  0E                                PUSH     CS
01E0  07                                POP      ES
01E1  8D 3E 0016 R                      LEA      DI,BPB                  ;ADDRESS OF BIOS PARAMETER BLOCK
01E5  83 C6 0B                          ADD      SI,11                   ;ADD 11 TO SI
01E8  B9 000D                           MOV      CX,13                   ;LENGTH OF BPB
01EB  F3/ A4                     REP    MOVSB
01ED  5B                                POP      BX                      ;RESTORE OFFSET OF SRH
01EE  07                                POP      ES                      ;RESTORE SEGMENT OF SRH
01EF  8D 16 0016 R                      LEA      DX,BPB                  ;GET BPB ARRAY POINTER
01F3  26: 89 57 12                      MOV      ES:BPBA_PTR[BX],DX      ;SAVE PTR TO BPB TABLE
01F7  26: 8C 4F 14                      MOV      ES:BPBA_PTR+2[BX],CS
01FB  26: 89 57 0E                      MOV      ES:DTA[BX],DX           ;OFFSET OF SECTOR BUFFER
01FF  26: 8C 4F 10                      MOV      ES:DTA+2[BX],CS
                                        STATUS   DONE,NOERROR,0
0203  26: 81 4F 03 0100     +            OR       ES:WORD PTR SRH_STA_FLD[BX],0100H
0209  26: 81 4F 03 0000     +            OR       ES:WORD PTR SRH_STA_FLD[BX],0
020F  EB 77 90                          JMP      EXIT
                                 ;
                                 ; THE FOLLOWING ENTRIES ARE FOR NOT SUPPORTED BY THIS DEVICE
                                 ;
0212                             IOCTL_IN:
0212                             IOCTL_OUT:
0212                             ND_INPUT:                               ;NON_DESTRUCTIVE INPUT NO WAIT (CHAR ONLY)
0212                             IN_STAT:                                ;INPUT STATUS                  "     "
0212                             IN_FLUSH:                               ;INPUT FLUSH                   "     "
0212                             OUT_STAT:                               ;OUTPUT STATUS                 "     "
0212                             OUT_FLUSH:                              ;OUTPUT FLUSH                  "     "
                                 ;
                                 ; DISK READ
                                 ;
0212                             INPUT:
0212  E8 0062 R                         CALL     IN_SAVE                 ;CALL THE INITIAL SAVE ROUTINE
0215  26: 8B 47 14                      MOV      AX,ES:WORD PTR SSN[BX]  ;SET STARTING SECTOR NUMBER
0219  2E: A3 0028 R                     MOV      CS:START_SEC,AX         ;SAVE STARTING SECTOR NUMBER
021D  26: 8B 47 12                      MOV      AX,ES:WORD PTR COUNT[BX]
0221  2E: A3 0025 R                     MOV      CS:TOTAL,AX             ;SAVE TOTAL SECTORS TO TRANSFER
0225  E8 00A3 R                         CALL     SECTOR_READ             ;READ IN THAT MANY SECTORS
0228  2E: 8B 1E 0012 R                  MOV      BX,CS:RH_OFF            ;RESTORE ES:BX AS REQUEST HEADER POINTER
022D  2E: 8E 06 0014 R                  MOV      ES,CS:RH_SEG
                                        STATUS   DONE,NOERROR,0          ;SET STATUS WORD (DONE, NOERROR)
0232  26: 81 4F 03 0100     +            OR       ES:WORD PTR SRH_STA_FLD[BX],0100H
0238  26: 81 4F 03 0000     +            OR       ES:WORD PTR SRH_STA_FLD[BX],0
023E  EB 48 90                          JMP      EXIT
                                 ;
                                 ; DISK WRITE
                                 ;
0241                             OUTPUT:                                 ;OUTPUT (WRITE)
0241  E8 0062 R                         CALL     IN_SAVE
0244  26: 8B 47 14                      MOV      AX,ES:WORD PTR SSN[BX]  ;GET STARTING SECTOR NUMBER
0248  2E: A3 0028 R                     MOV      CS:START_SEC,AX         ;SET    "       "      "
024C  26: 8B 47 12                      MOV      AX,ES:WORD PTR COUNT[BX]
0250  2E: A3 0025 R                     MOV      CS:TOTAL,AX             ;SAVE TOTAL SECTORS TO WRITE
0254  E8 00C0 R                         CALL     SECTOR_WRITE            ;WRITE OUT THOSE SECTORS
```

```
0257  2E: 8B 1E 0012 R                    MOV     BX,CS:RH_OFF            ;RESTORE ES:BX AS REQUEST HEADER POINTER
025C  2E: 8E 06 0014 R                    MOV     ES,CS:RH_SEG
0261  2E: 80 3E 0027 R 00                 CMP     CS:BYTE PTR VERIFY,0    ;WRITE VERIFY SET
0267  74 08                               JZ      NO_VERIFY               ;NO, NO WRITE VERIFY
0269  2E: C6 06 0027 R 00                 MOV     CS:BYTE PTR VERIFY,0    ;RESET VERIFY INDICATOR
026F  EB A1                               JMP     INPUT                   ;READ THOSE SECTORS BACK IN
0271                              NO_VERIFY:
                                          STATUS  DONE,NOERROR,0          ;SET DONE, NO ERROR IN STATUS WORD
0271  26: 81 4F 03 0100       +            OR      ES:WORD PTR SRH_STA_FLD[BX],0100H
0277  26: 81 4F 03 0000       +            OR      ES:WORD PTR SRH_STA_FLD[BX],0
027D  EB 09 90                            JMP     EXIT
0280                              OUT_VERIFY:                             ;OUTPUT (WRITE) WITH VERIFY
0280  2E: C6 06 0027 R 01                 MOV     CS:BYTE PTR VERIFY,1    ;SET THE VERIFY FLAG
0286  EB B9                               JMP     OUTPUT                  ;BRANCH TO OUTPUT ROUTINE
                                  ;
                                  ; COMMON EXIT
                                  ;
0288                              EXIT:
0288  5E                                  POP     SI                      ;RESTORE ALL OF THE REGISTERS
0289  5F                                  POP     DI
028A  5A                                  POP     DX
028B  59                                  POP     CX
028C  5B                                  POP     BX
028D  58                                  POP     AX
028E  07                                  POP     ES
028F  1F                                  POP     DS
0290  CB                                  RET
0291                              E_O_P:
                                  ; MACRO TO ALIGN THE VIRTUAL DISK ON A PARAGRAPH BOUNDARY
                                  if ($-START) MOD 16
02A0                              ORG     ($-START)+16-(($-START) MOD 16)
                                  endif
 = 02A0                           VDISK   EQU     $
02A0                              VDSK    ENDP
02A0                              CSEG    ENDS
                                          END     BEGIN
```